
Subject: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID

Posted by [Pierre Peiffer](#) on Tue, 29 Jan 2008 16:02:38 GMT

[View Forum Message](#) <> [Reply to Message](#)

From: Pierre Peiffer <pierre.peiffer@bull.net>

This patch provides three new API to change the ID of an existing System V IPCs.

These APIs are:

```
long msg_chid(struct ipc_namespace *ns, int id, int newid);
long sem_chid(struct ipc_namespace *ns, int id, int newid);
long shm_chid(struct ipc_namespace *ns, int id, int newid);
```

They return 0 or an error code in case of failure.

They may be useful for setting a specific ID for an IPC when preparing a restart operation.

To be successful, the following rules must be respected:

- the IPC exists (of course...)
- the new ID must satisfy the ID computation rule.
- the entry in the idr corresponding to the new ID must be free.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>

Acked-by: Serge Hallyn <serue@us.ibm.com>

```
include/linux/msg.h | 2 ++
include/linux/sem.h | 2 ++
include/linux/shm.h | 3 +++
ipc/msg.c           | 45 +++++
ipc/sem.c           | 51 +++++
ipc/shm.c           | 45 +++++
ipc/util.c          | 48 +++++
ipc/util.h          | 1 +
8 files changed, 197 insertions(+)
```

Index: b/include/linux/msg.h

=====

--- a/include/linux/msg.h

+++ b/include/linux/msg.h

@@ -63,6 +63,7 @@ struct msginfo {

#ifdef __KERNEL__

#include <linux/list.h>

+#include <linux/ipc_namespace.h>

```

/* one msg_msg structure for each message */
struct msg_msg {
@@ -96,6 +97,7 @@ extern long do_msgsnd(int msqid, long mt
    size_t msgsz, int msgflg);
extern long do_msgrcv(int msqid, long *pmtyp, void __user *mtext,
    size_t msgsz, long msgtyp, int msgflg);
+long msg_chid(struct ipc_namespace *ns, int id, int newid);

#endif /* __KERNEL__ */

```

Index: b/include/linux/sem.h

```

=====
--- a/include/linux/sem.h
+++ b/include/linux/sem.h
@@ -138,9 +138,11 @@ struct sysv_sem {
};

#ifdef CONFIG_SYSVIPC
+#include <linux/ipc_namespace.h>

extern int copy_semundo(unsigned long clone_flags, struct task_struct *tsk);
extern void exit_sem(struct task_struct *tsk);
+long sem_chid(struct ipc_namespace *ns, int id, int newid);

```

```

#else
static inline int copy_semundo(unsigned long clone_flags, struct task_struct *tsk)
Index: b/include/linux/shm.h

```

```

=====
--- a/include/linux/shm.h
+++ b/include/linux/shm.h
@@ -104,8 +104,11 @@ struct shmid_kernel /* private to the ke
#define SHM_NORESERVE 010000 /* don't check for reservations */

#ifdef CONFIG_SYSVIPC
+#include <linux/ipc_namespace.h>
+
long do_shmat(int shmid, char __user *shmaddr, int shmflg, unsigned long *addr);
extern int is_file_shm_hugepages(struct file *file);
+long shm_chid(struct ipc_namespace *ns, int id, int newid);
#else
static inline long do_shmat(int shmid, char __user *shmaddr,
    int shmflg, unsigned long *addr)
Index: b/ipc/msg.c

```

```

=====
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -291,6 +291,51 @@ asmlinkage long sys_msgget(key_t key, in

```

```

    return ipcget(ns, &msg_ids(ns), &msg_ops, &msg_params);
}

/* must be called with mutex and msq locks held */
+static long msg_chid_nolock(struct ipc_namespace *ns, struct msg_queue *msq,
+    int newid)
+{
+    long err;
+
+    err = ipc_chid(&msg_ids(ns), msq->q_perm.id, newid);
+    if (!err)
+        msq->q_ctime = get_seconds();
+
+    return err;
+}
+
+/* API to use for changing an id from kernel space, not from the syscall, as
+    there is no permission check done here */
+long msg_chid(struct ipc_namespace *ns, int id, int newid)
+{
+    long err;
+    struct msg_queue *msq;
+
+    retry:
+    err = idr_pre_get(&msg_ids(ns).ipcs_idr, GFP_KERNEL);
+    if (!err)
+        return -ENOMEM;
+
+    down_write(&msg_ids(ns).rw_mutex);
+    msq = msg_lock_check(ns, id);
+
+    if (IS_ERR(msq)) {
+        up_write(&msg_ids(ns).rw_mutex);
+        return PTR_ERR(msq);
+    }
+
+    err = msg_chid_nolock(ns, msq, newid);
+
+    msg_unlock(msq);
+    up_write(&msg_ids(ns).rw_mutex);
+
+    /* ipc_chid may return -EAGAIN in case of memory requirement */
+    if (err == -EAGAIN)
+        goto retry;
+
+    return err;
+}
+

```

```
static inline unsigned long
copy_msqid_to_user(void __user *buf, struct msqid64_ds *in, int version)
{
```

Index: b/ipc/sem.c

```
=====
```

```
--- a/ipc/sem.c
```

```
+++ b/ipc/sem.c
```

```
@@ -564,6 +564,57 @@ static void freeary(struct ipc_namespace
    ipc_rcu_putref(sma);
}
```

```
/* must be called with rw_mutex and sma locks held */
```

```
+static long sem_chid_nolock(struct ipc_namespace *ns, struct sem_array *sma,
```

```
+    int newid)
```

```
+{
```

```
+ long err;
```

```
+
```

```
+ err = ipc_chid(&sem_ids(ns), sma->sem_perm.id, newid);
```

```
+
```

```
+ if (!err) {
```

```
+ struct sem_undo *un;
```

```
+ for (un = sma->undo; un; un = un->id_next)
```

```
+ un->semid = newid;
```

```
+
```

```
+ sma->sem_ctime = get_seconds();
```

```
+ }
```

```
+
```

```
+ return err;
```

```
+}
```

```
+
```

```
/* API to use for changing an id from kernel space, not from the syscall, as
```

```
+ there is no permission check done here */
```

```
+long sem_chid(struct ipc_namespace *ns, int id, int newid)
```

```
+{
```

```
+ long err;
```

```
+ struct sem_array *sma;
```

```
+
```

```
+retry:
```

```
+ err = idr_pre_get(&sem_ids(ns).ipcs_idr, GFP_KERNEL);
```

```
+ if (!err)
```

```
+ return -ENOMEM;
```

```
+
```

```
+ down_write(&sem_ids(ns).rw_mutex);
```

```
+ sma = sem_lock_check(ns, id);
```

```
+
```

```
+ if (IS_ERR(sma)) {
```

```
+ up_write(&sem_ids(ns).rw_mutex);
```

```
+ return PTR_ERR(sma);
```

```

+ }
+
+ err = sem_chid_nolock(ns, sma, newid);
+
+ sem_unlock(sma);
+ up_write(&sem_ids(ns).rw_mutex);
+
+ /* ipc_chid may return -EAGAIN in case of memory requirement */
+ if (err == -EAGAIN)
+ goto retry;
+
+ return err;
+}
+
static unsigned long copy_semid_to_user(void __user *buf, struct semid64_ds *in, int version)
{
    switch(version) {
Index: b/ipc/shm.c
=====
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -162,7 +162,52 @@ static inline int shm_addid(struct ipc_n
    return ipc_addid(&shm_ids(ns), &shp->shm_perm, ns->shm_ctlmni);
}

+/* must be called with mutex and shp locks held */
+static long shm_chid_nolock(struct ipc_namespace *ns, struct shmid_kernel *shp,
+    int newid)
+{
+    long err;
+
+    err = ipc_chid(&shm_ids(ns), shp->shm_perm.id, newid);
+    if (!err) {
+        shp->shm_file->f_dentry->d_inode->i_ino = newid;
+        shp->shm_ctim = get_seconds();
+    }
+
+    return err;
+}
+
+/* API to use for changing an id from kernel space, not from the syscall, as
+    there is no permission check done here */
+long shm_chid(struct ipc_namespace *ns, int id, int newid)
+{
+    long err;
+    struct shmid_kernel *shp;
+
+retry:

```

```

+ err = idr_pre_get(&shm_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+   return -ENOMEM;
+
+ down_write(&shm_ids(ns).rw_mutex);
+ shp = shm_lock_check(ns, id);
+
+ if (IS_ERR(shp)) {
+   up_write(&shm_ids(ns).rw_mutex);
+   return PTR_ERR(shp);
+ }
+
+ err = shm_chid_nolock(ns, shp, newid);

+ shm_unlock(shp);
+ up_write(&shm_ids(ns).rw_mutex);
+
+ /* ipc_chid may return -EAGAIN in case of memory requirement */
+ if (err == -EAGAIN)
+   goto retry;
+
+ return err;
+}

```

/* This is called by fork, once for every shm attach. */

static void shm_open(struct vm_area_struct *vma)

Index: b/ipc/util.c

=====

--- a/ipc/util.c

+++ b/ipc/util.c

@@ -363,6 +363,54 @@ retry:

```

/**
+ * ipc_chid - change an IPC identifier
+ * @ids: IPC identifier set
+ * @oldid: ID of the IPC permission set to move
+ * @newid: new ID of the IPC permission set to move
+ *
+ * Move an entry in the IPC idr from the 'oldid' place to the
+ *   'newid' place. The seq number of the entry is updated to match the
+ *   'newid' value.
+ *
+ * Called with the ipc lock and ipc_ids.rw_mutex held.
+ */
+int ipc_chid(struct ipc_ids *ids, int oldid, int newid)
+{
+ struct kern_ipc_perm *p;

```

```

+ int old_lid = oldid % SEQ_MULTIPLIER;
+ int new_lid = newid % SEQ_MULTIPLIER;
+
+ if (newid != (new_lid + (newid/SEQ_MULTIPLIER)*SEQ_MULTIPLIER))
+ return -EINVAL;
+
+ p = idr_find(&ids->ipcs_idr, old_lid);
+
+ if (!p)
+ return -EINVAL;
+
+ /* The idx in the idr may be the same but not the seq number. */
+ if (new_lid != old_lid) {
+ int id, err;
+
+ err = idr_get_new_above(&ids->ipcs_idr, p, new_lid, &id);
+ if (err)
+ return err;
+
+ /* do we get our wished id ? */
+ if (id == new_lid) {
+ idr_remove(&ids->ipcs_idr, old_lid);
+ } else {
+ idr_remove(&ids->ipcs_idr, id);
+ return -EBUSY;
+ }
+ }
+
+ p->id = newid;
+ p->seq = newid/SEQ_MULTIPLIER;
+ return 0;
+}
+
+/**

```

* ipc_rmid - remove an IPC identifier

* @ids: IPC identifier set

* @ipcp: ipc perm structure containing the identifier to remove

Index: b/ipc/util.h

=====

--- a/ipc/util.h

+++ b/ipc/util.h

@@ -85,6 +85,7 @@ int ipc_get_maxid(struct ipc_ids *);

void ipc_rmid(struct ipc_ids *, struct kern_ipc_perm *);

/* must be called with ipcp locked */

+int ipc_chid(struct ipc_ids *ids, int oldid, int newid);

int ipcperms(struct kern_ipc_perm *ipcp, short flg);

/* for rare, potentially huge allocations.

--

Pierre Peiffer

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
